Inicialização:

```
1  .data
2
3  input: .asciiz "Digite o ângulo em graus cos(x): "
4  output_1: .asciiz "cos("
5  output_2: .asciiz ") = : "
6  pi: .float 3.14159265359
7  pi_graus: .float 180.0
```

Na linha 3, 4 e 5 as variáveis servem apenas para interagir com o usuário. Enquanto na linha 6 e 7 são os valores necessários para fazer a conversão de graus para radianos.

Main do programa:

Até a linha 20 o programa irá interagir com o usuário e recolher o ângulo que for definido na leitura. Na linha 21 e 22, respectivamente, há a passagem do parâmetro para a função e a própria função ‘conversor’. O objetivo dessa função é converter o ângulo informado em graus para radianos.

Após isso, na linha 27, a função chamada será a mais importante do programa, a qual será responsável por calcular o cosseno a partir do Polinômio de Taylor.

Enfim, da linha 29 em diante o computador irá informar ao usuário qual foi o resultado obtido.

```
10
11 .main:
12         li $v0, 4
13         la $a0, input
14         syscall
15
16         li $v0, 6
17         syscall
18
19         mov.s $f13, $f0
20
21         mfc1 $a0, $f0
22         jal conversor
23
24         mov.s $f11, $f12
25
26         mfc1 $a0, $f11
27         jal taylor
28
29         li $v0, 4
30         la $a0, output_1
31         syscall
32
33         li $v0, 2
34         mov.s $f12, $f13
35         syscall
36
37         li $v0, 4
38         la $a0, output_2
39         syscall
40
41         li $v0, 2
42         mov.s $f12, $f0
43         syscall
44
45         li $v0, 10
46         syscall
```

Função ‘taylor’:

Inicialmente, o registrador $f0 (L50) será o responsável por guardar a Série de Taylor, assim sendo inicializado com o valor 0. Em seguida o loop criado (54) deverá repetir-se 8 vezes, uma vez que foi definido os 8 primeiros termos para compôr o cálculo.

No loop, a primeira coisa feita é chamar a função potencia (59), a qual recebe dois argumentos, $a0 (58) e $f11 (57), sendo aquele o responsável por informar qual o grau da potência na ordem k * 2, além de já aproveitar e utilizar o registrador responsável por iterar essa repetição $t6 (51), e este sendo o valor que foi digitado pelo usuário convertido em radianos.

Logo mais, a função fatorial (63) é chamada na ordem k * 2, também com o $t6 (61). Consequentemente, obtém-se os dois valores necessários para descobrir cada termo da série. Efetuando-se a divisão (68) em seguida.

Na parte final, deve-se observar se o termo deve ser somado ou subtraído da série, utilizando-se do resto da divisão do iterador / 2 (70, 71). Assim a função retorna para a main.

```
48  taylor:
49
50          mtc1 $zero, $f0 # soma dos valores
51          li $t6, -1
52          move $t7, $ra
53
54          loop: # repetirá 7 vezes
55                  addi $t6, $t6, 1
56
57                  mfc1 $a1, $f11
58                  move $a0, $t6
59                  jal potencia
60
61                  move $a0, $t6
62                  mul $a0, $a0, 2
63                  jal fatorial
64
65                  mtc1 $v0, $f9
66                  cvt.s.w $f9, $f9
67
68                  div.s $f12, $f10, $f9
69
70                  div $t5, $t6, 2
71                  mfhi $t5
72
73                  beqz $t5, par
74
75                  sub.s $f0, $f0, $f12
76                  j fim_loop
77
78                  par:
79                          add.s $f0, $f12, $f0
80
81                  fim_loop:
82                  blt $t6, 7, loop
83
84          mfc1 $v0, $f0
85
86          jr $t7
87
```

Função ‘potencia’:

Essa função começa multiplicando (90) o argumento do iterador por 2, uma vez que as potências são todas múltiplas de 2. Logo em seguida haverá a estrutura de repetição ‘potencia_repete’ (95) que irá repetir de acordo com o resultado anterior de i * 2, atualizando o valor multiplicando-se por si mesmo (97).

Também é importante mencionar o rótulo ‘potencia_0’ (108) que se responsabiliza por retornar o número 1 caso o argumento que corresponde à ordem da potência vier igual a 0.

```
potencia:

        mul $a0, $a0, 2 # potencia
        mov.s $f10, $f11
        li $t0, 0

        beqz $a0, potencia_0
        potencia_repete:

                mul.s $f10, $f10, $f11

                addi $t0, $t0, 1
                move $t2, $t0
                addi $t2, $t2, 1

                bne $t2, $a0, potencia_repete

        mfc1 $v0, $f10
        j sair_potencia

        potencia_0:

                li $t2, 1
                mtc1 $t2, $f10
                cvt.s.w $f10, $f10
                mfc1 $v0, $f10

        sair_potencia:

        jr $ra
```

Função ‘conversor’:

```
conversor:

        lwc1 $f1, pi # carrega pi para $f0
        lwc1 $f2, pi_graus # carrega 180 para #f2

        mul.s $f12, $f0, $f1 # multiplica a entrada por pi
        div.s $f12, $f12, $f2 # divide o resultado por 180

        mfc1 $v0, $f12 # move o resultado para saída da função

        jr $ra
```

Essa função basicamente carrega (121, 122) o valor das variáveis definidas no .data e faz a conversão $\frac{\alpha \cdot \pi}{180^{o}}$, no qual α é o ângulo digitado pelo usuário.

Função ‘fatorial’:

```
131 fatorial:
132
133         addiu $sp, $sp, -8      # ajusta a pilha para receber 2 itens
134         sw    $ra, 4($sp)       # salva o endereço de retorno
135         sw    $a0, 0($sp)       # salva o argumento da função
136
137         bne   $zero, $a0, n_nao_igual_0 # se n!=0  calcule n*fatorial(n-1)
138
139 n_igual_0:
140
141         add   $v0, $zero, 1     # retorna 1 = 0!
142         j fatorial_epilogo      # epílogo  do procedimento
143
144 n_nao_igual_0:
145
146         addi  $a0, $a0, -1      # a0 <- n-1
147         jal   fatorial          # chamamos fatorial(n-1)
148         lw    $a0, 0($sp)       # a0 <- n, restauramos n
149         mul   $v0, $a0, $v0     # v0 <- n*fatorial(n-1), v0 valor de retorno
150         lw    $ra, 4($sp)       # restaura o endereço de retorno
151
152 fatorial_epilogo:
153
154         add   $sp, $sp, 8       # restaura a pilha - eliminamos 2 itens
155         jr    $ra               # retorna para o procedimento chamador
156
```

Enfim, essa função calcula o fatorial do número passado pelo registrador $a0 (135) de uma forma recursiva. No rótulo ‘n_igual_0’, a função irá retornar 1, pois ele é o responsável pelo 0!. Enquanto o ‘n_nao_igual_0’ (144) é quem chama a própria função (147) novamente para consumir a pilha.

Resultado:

O da esquerda é o valor encontrado no código, enquanto o da direita é o valor calculado pela calculadora. É possível notar a proximidade dos dois números, entretanto ainda há uma certa diferença, pois o Polinômio de Taylor está funcionando até n = 8.